# CS 470 Project Two Script Template

John Brungard | CS470 | Project Presentation | 12/2023 | Link: https://youtu.be/AW7lhsrwh\_U?feature=shared

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is John Brungard and welcome to my presentation! The presentation covers the process of migrating a full stack application to an AWS serverless solution. |
| **2** | The purpose of this presentation is to articulate the intricacies of cloud development to both technical and nontechnical audiences. I hope you walk away with more knowledge on AWS. |
| **3** | The model for migrating a full stack application to the cloud involves the 5 R’s which consist of refactor, replatform, repurchase, rehost, and relocate. Refactoring is making changes to the application’s architecture to leverage more uses of Cloud Capabilities. Replatforming is transferring the application to the cloud and incorporating some optimization for cloud services. Repurchase is transitioning to another product for the cloud by adopting a Software as a Service model. Rehost is migrating an application as is into a cloud environment. Relocate is shifting infrastructure to the cloud without needing new hardware, or adjustments to current operations.  The tools necessary for containerization is Docker and MongoDB. Docker is a Platform as a service product that delivers software in packages as containers. This allows building, testing, and deploying applications more quickly. The other tool is MongoDB, which is the JSON formatted database where you can hold data stored within a MongoDB container. In unison, this allows users to create a portable NOSQL database. |
| **4** | What is the value behind Docker Compose? Well, the big advantage of using Compose is you can define your application stack in a file, keep it at the root of your project repository (it's now version controlled), and easily enable someone else to contribute to your project. Someone would only need to clone your repository and start the app using Compose. |
| **5** | Docker Compose uses a Yet Another Markup Language (YAML) file to define the services, network, and storage volumes that an application will use. The image shows this by its configuration of the angular front-end application using port 4200 as the local host and running the commands npm install and ng serve. |
| **6** | Now we look at the serverless cloud. To define serverless, I would say it is a cloud-native development model that allows developers to build and run applications without having to manage servers. Serverless still does have servers, but their complexities are abstracted from development. Serverless Applications will scale automatically as a user base or usage increases. This is an advantage over traditional applications that can be burdened quickly with an increase in usage. Serverless applications also are not hosted on the original server and can potentially run on servers closer to the end user which can reduce latency. Finally, there is no need for uploads of code to servers. This makes it easier and quicker to update, patch, deploy and fix parts of an application.  Now let’s look at S3 Storage. S3 is a storage that provides object storage through web service interface. When local disc storage crashes or the server associated with it crashes, users do not have access to their files. If a server crashes with AWS S3, you can access your files by running on a backup server and connect to the bucket using your S3 login credentials. Also, the AWS S3 will likely encounter more latency during file transfers than local storage. This is because after files are uploaded to the server, they still must be forwarded to S3 infrastructure, making a longer path.  The accompanying image shows the art of an application being serverless using AWS Lambda. In this model, the user takes less responsibility for provisioning as the business logic is fed to a serverless solution like lambda and provisioning is handled by the provider. While the static content of the S3 bucket is not serverless, the handling of app data, api keys and identities are made of instances that are indeed serverless. This means DynamoDB, STS, and Cognito are not provisioned by the user. |
| **7** | There are many advantages of using a serverless API. Serverless APIs are very scalable as every function be handled and scaled independently. Serverless is also cheap, consumes no resources at rest and puts most of the costs on the provider instead of the user. Serverless APIs also can have improved geolocation as servers are created on demand, which means a function can be run in an environment close to a user’s request to reduce latency.  Now let’s investigate Lambda API logic. AWS Lambda executes code in response to AWS service events such as adding/deleting files in an S3 bucket, making an HTTP request to the Amazon API gateway, and so on. Essentially, every Lambda function has a container of its own. When functions are written, Lambda puts it into its own container and it is executed on the multi-tenant cluster of machines managed by AWS, which also manages the infrastructure layer of AWS Lambda.  There are some steps needed to integrate the frontend with the backend. First, we must create a backend and frontend. Next, we need to make our lambda functions and follow up with populating and linking our NOSQL database. Then we configure an API using API Gateway and implement CORs. Finally, we deploy the API.  The image shown shows a summary of the AWS Cloud. End users communicate with API Gateway using method and integration requests which in turn uses a REST API with AWS Lambda using common commands such as GET, POST, and PUT. These commands are used in conjunction with DynamoDB to retrieve and show contents of the database back to the user. |
| **8** | There are some notable differences when looking at our NOSQL databases MongoDB and DynamoDB. To start, MongoDB can query using single keys, ranges, faceted search, JOINs, graph traversals, and geospatial queries. DynamoDB uses key-value queries only and can have at most 2 attributes. For indexing, secondary indexes can be defined on any field at any time for MongoDB. For DynamoDB, local secondary indexes must be defined when the table is created. MongoDB also has query backups that let you search backups without restoring data while DynamoDB does not offer this luxury.  For the project, queries for the question included a unique ID in which to retrieve from the database, a category and question slug, the question itself, the negative and positive votes for each question, and an empty array for the possible answers. Queries for the answer included the answer itself, any positive and negative votes, a unique ID in which to retrieve it from the database, and the question ID to find the corresponding question to the answer. |
| **9** | For Cloud-Based Development Principles, we look at elasticity and pay-for-use models. Elasticity is the ability to manipulate the resources a cloud-based application uses. The Pay-for-use is a model that does not assign a fixed fee per a certain time iteration. You pay for what resources or compute power you use.  In our graph we see how the y axis is assigned compute power and our x axis is assigned time. Clusters are monitored and scaled horizontally and vertically as needed as seen in the blue line representing planned capacity. The red line is the actual usage. We also see how the application can be scaled to meet demand and then scale back when demand is low. This is because we are transforming the infrastructure model into one that can dynamically adapt with an Elastic Cloud Infrastructure. |
| **10** | Let’s look next into securing our Cloud App. We prevent unauthorized access using methods such as encryption, tokenization, data decomposition, and cyber deception.  Encryption makes data unreadable by unauthorized users or those who do not have an encryption key.  Tokenization is a process that allows you to define a sequence of data to represent an otherwise sensitive piece of information (e.g., a token to represent a customer’s credit card number).  Data decomposition is a process in which data sets are turned into unrecognized elements that have no resemblance to the original data.  Finally, Cyber Deception is using sophisticated traps and decoys to trick unauthorized users in thinking they have infiltrated the system when they are being monitored and studied to prevent future attempts and to stop the current attempt. |
| **11** | Next let’s look at the differences between roles and policies. The difference between IAM roles and policies in AWS is that a role is a type of IAM identity that can be authenticated and authorized to utilize an AWS resource, whereas a policy defines the permissions of the IAM identity.  The second piece of information involves custom policies, but this was not applicable to our application. |
| **12** | Let’s now investigate API security. To secure connections between lambda and gateway, you can use an HTTP(S) endpoint that your Lambda function can connect to. This allows the API Gateway to forward requests to the corresponding database and work in correlation with Lambda function(s). You can also use IAM permissions to control access for invoking an API.  To secure connections between Lambda and the database, you can use a Virtual Private Cloud (VPC) endpoint if the database is hosted in the same VPC as your Lambda function(s). This can improve security and help with connection interruptions. You can also use a connection pooling library to manage database connections.  To secure the S3 bucket, we use s3 block public access, disable ACLs and implement the principle of least privilege. |
| **13** | With AWS Identity and Access Management, you can select what or who can access the services and resources within your AWS. You can also manage permissions and analyze the access to refine permissions across AWS. This is a very good starting point for securing your cloud application. |
| **14** | To conclude, I would like to leave the presentation on three main ideas.  One, use one or more of the following techniques to prevent unauthorized access: encryption, tokenization, data decomposition, and cyber deception.  Two, a serverless API doesn’t mean the application doesn’t have servers. Rather, it means the provisioning is done by the provider instead of the user. This comes with added benefits of scalability, cost (pay-for-use) and the potential for an improved geolocation.  Three, S3 is a storage that provides object storage through web service interface. While it may encounter more latency, users can still retrieve their data when a server crashes unlike local storage.  What are your questions involving the presentation? \*\*\*Brief silence\*\*\*  Thank you for your time! |
| **15** | This last slide shows the references used to create and conduct this presentation. |